Cyber Security
Identification, authentication, authorization
Grégoire Payen de La Garanderie
Access Control

Access to the resources:
- Claim the identity
- Verify the credentials
- Check permissions
- Grant access

Identification → Authentication → Authorization
Terminology

- **Subject** — An active entity within a system (physical person, script, etc)
- **Principal** — An entity that can be granted access (represented by a username, userid, pin, etc)
Identification

The subject (a person, a script, etc) identifies itself to the system as a principal (represented by a username, userid, pin, etc).

Say your name

Hi I’m John

Enter your username

john.doe

Present yourself

😊
The system verifies the identity of the user.

Present credentials

Enter your password

Scan your fingerprints

john.doe

**************************
Authorization

Terminology

- **Subject** — An active entity within a system (physical person, script, etc)
- **Principal** — An entity that can be granted access (represented by a username, userid, pin, etc)
- **Object** — Resource that some principals may access or use

Permissions (e.g. Unix)
-rwxr-x--- johndoe compsci

Access lists (e.g. Apache)
<RequireAll>
  Require all granted
  Require not ip 10.252.46.165
</RequireAll>
Authorization

The system checks that the principal has the permissions to access an object.

Permissions (e.g. Unix)
-rwxr-x--- johndoe compsci

Access lists (e.g. Apache)
<pre><code>&lt;RequireAll&gt;
  Require all granted
  Require not ip 10.252.46.165
&lt;/RequireAll&gt;</code></pre>
Credentials

Terminology

● What you know?
  ○ Passwords, pin numbers
● What you have?
  ○ Authentication key, passport, ticket, mobile phone
● Who you are?
  ○ Biometrics (fingerprints, DNA, face recognition)
# The problem with Passwords

The top 20 most popular passwords (ranking 2018)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td><strong>123456</strong> (Unchanged)</td>
</tr>
<tr>
<td>2.</td>
<td><strong>Password</strong> (Unchanged)</td>
</tr>
<tr>
<td>3.</td>
<td><strong>123456789</strong> (Up 3)</td>
</tr>
<tr>
<td>4.</td>
<td><strong>12345678</strong> (Down 1)</td>
</tr>
<tr>
<td>5.</td>
<td><strong>12345</strong> (Unchanged)</td>
</tr>
<tr>
<td>6.</td>
<td><strong>111111</strong> (New)</td>
</tr>
<tr>
<td>7.</td>
<td><strong>1234567</strong> (Up 1)</td>
</tr>
<tr>
<td>8.</td>
<td><strong>sunshine</strong> (New)</td>
</tr>
<tr>
<td>9.</td>
<td><strong>qwerty</strong> (Down 5)</td>
</tr>
<tr>
<td>10.</td>
<td><strong>iloveyou</strong> (Unchanged)</td>
</tr>
<tr>
<td>11.</td>
<td><strong>princess</strong> (New)</td>
</tr>
<tr>
<td>12.</td>
<td><strong>admin</strong> (Down 1)</td>
</tr>
<tr>
<td>13.</td>
<td><strong>welcome</strong> (Down 1)</td>
</tr>
<tr>
<td>14.</td>
<td><strong>666666</strong> (New)</td>
</tr>
<tr>
<td>15.</td>
<td><strong>abc123</strong> (Unchanged)</td>
</tr>
<tr>
<td>16.</td>
<td><strong>football</strong> (Down 7)</td>
</tr>
<tr>
<td>17.</td>
<td><strong>123123</strong> (Unchanged)</td>
</tr>
<tr>
<td>18.</td>
<td><strong>monkey</strong> (Down 5)</td>
</tr>
<tr>
<td>19.</td>
<td><strong>654321</strong> (New)</td>
</tr>
<tr>
<td>20.</td>
<td><strong>!@#$%^&amp;</strong>* (New)</td>
</tr>
</tbody>
</table>

Source: https://bit.ly/2Cq3O8e

123456 has been used by almost 3% of people.
The problem with Passwords

Common Security Guidelines

- Adopt long passphrases
- Avoid easy to guess passwords
- Use combination of a-z, A-Z, 0-9 and symbols
- Do not write down passwords
- Avoid using the same password for multiple services

However — when Internet users log on to as many as 25 password-protected sites per day, remembering a different 14-character password is a Herculean mental exercise.
The problem with Passwords

Never write down plain passwords
But store them in a password-protected password agent

E.g.
Websites: LastPass, DashLane, 1Password
Tools: Roboform, PasswordSafe,
Keepass (Windows), Keepassxc (Linux)
Authentication keys (e.g. SSH keys)

- Similar to passwords, but
- RSA-based keys
- Subject create private/public key
- Share the public key with services
- Per device RSA key
Authentication keys

Advantages

● Public key leak are inconsequential
● Compromised device access can be revoked
Authentication keys

Demo time: SSH keys

Setting up SSH keys for your client

ssh-keygen
ssh-copy-id xzpq33@mira.dur.ac.uk
ssh xzpq33@mira.dur.ac.uk

Public key: .ssh/id_rsa.pub
Private key: .ssh/id_rsa
Server authorized keys: .ssh/authorized_keys

(create key for your client)
(send public key to server)
(ssh'ing using your key)
(publicly share)
(do not share)
(server side)
Security Keys

Authentication keys weakness: Compromised client

Solution: Physical security keys

- Static password token
  (not recommended)
- Asynchronous tokens
  (one-time passwords)
- Challenge-response tokens

One time password: Yubikey

One time password: For a well known bank

Car keys
Biometrics

History of fingerprint

1982 Ink & paper
1990s Optical
1990s Capacitive
1997 First swip sensor
2007 Slap sensor
2010 Touchless swipe sensor
Biometrics

- Fingerprints
- Iris recognition
- DNA matching
- Face recognition
- Keystroke & Mouse biometrics
- Gait recognition
Biometrics

Advantages

Non-repudiation: a way to guarantee that an individual who accesses a certain facility cannot later deny using it

Disadvantages

Uncertainty: Compromise between false-positives and false-negatives.
Receiver Operating Characteristic (ROC) curve

FMR = number of false positives / total matches

FNMR = number of false negatives / total matches
Performance Policy

- Prefer low FMR
  E.g. automatic border control
  Refer to human on negative result

- Prefer low FNMR
  E.g. suspect recognition on CCTV
  Refer to human on positive result
Two-factor authentication

Combine two authentication factors from:

- "What you know": password, pin
- "What you have": mobile phone, authentication key

Best Practice!
Example: Bitcoin

- Principal: Public key
- Authentication factor: Public/private key

- Authorization mechanism:
  each object (transaction) output has an associated script controlling permissions:

Standard transaction (pay-to-pubkey-hash):
  scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
  scriptSig: <sig> <pubKey>

Funds freezed until specified time:
  scriptPubKey: <expiry time> OP_CHECKLOCKTIMEVERIFY OP_DROP OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
  scriptSig: <sig> <pubKey>
Example: Bitcoin

Standard transaction (pay-to-pubkey-hash):
scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG

scriptSig: <sig> <pubKey>

Funds freezed until specified time:
scriptPubKey: <expiry time> OP_CHECKLOCKTIMEVERIFY
OP_DROP OP_DUP OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG

scriptSig: <sig> <pubKey>
Zero-knowledge Password Proof (ZKPP)

Objective: Do not reveal anything in the client/server communications about the password
Otherwise we are vulnerable to replay attacks.

Most common ZKPP approach: Challenge-response auth

- Server generate unique challenge value: nonce
- Server send nonce to the client
- Client compute response = hash(nonce + password)
- Client send response back to server
- Server compare the response with hash(nonce + stored password)

More info: SCRAM-SHA-256 authentication
Zero-knowledge Password Proof (ZKPP)

Nonce Properties:

- Nonce: Random or pseudo-random unique value
- Uniqueness: Prevent replay attacks
- Susceptible to PRNG flaws: Such as the Dual_EC_DRBG “potential backdoor”
Example: EMV payment

Standard used by all credit cards
EMV standard: Initially written in 1993
Over 3600 pages of protocol specification
Requirements varies from bank to banks

Protocol evolution as attacks gets more sophisticated.
Example: EMV payment

Card authentication mechanism

- Static data authentication (offline)
- Dynamic data authentication (offline)
- Combined DDA with application cryptogram generation (offline)
- Cryptogram (online)

Multiple cardholder verification mechanism (CVM):

- No CVM required (e.g. motorway toll)
- Signature (common in some countries, e.g. US)
- Offline PIN (no internet, pin verified by the card)
- Online PIN (internet, pin verified by the bank)
Example: EMV payment

SDA: Static Data Authentication

Offline card payment
Used by old card & terminal
Lowest common denominator

Vulnerable to skimming attack

During transaction, terminal records the static data

A cloned card is created with the same static data.
Example: EMV payment

SDA: Static Data Authentication

Card → Terminal → User

- records, $\text{sig}_{\text{BANK}}\{\text{records}\}$ → PIN
- PIN OK/WRONG
- $\text{transaction, nonce}$
- MAC($\text{transaction, PIN OK}$)

* MAC: Message Authentication Code, computed by the card from a unique key
Example: EMV payment

SDA: Static Data Authentication & yes card attack

An attacker can get records, $\text{sig}_{\text{BANK}}\{\text{records}\}$ by listening to a valid transaction.

Then the attacker can create a fake card using $\text{sig}_{\text{BANK}}\{\text{records}\}$ and generate an invalid MAC. For offline transaction, the merchant cannot verify the MAC anyway. By the time the merchant send the transactions to the bank, the attacker will be long gone!

Problem: Static password
Example: EMV payment

SDA: Static Data Authentication

Fake Card | Terminal | User
--- | --- | ---
records, $\text{sig}_{\text{BANK}}\{\text{records}\}$ | Ask PIN | 
PIN | Enter PIN | 
PIN OK | 
transaction, nonce | 
Invalid MAC | 

* MAC: Message Authentication Code, computed by the card from a unique key
Example: EMV payment

DDA: Dynamic Data Authentication

Use challenge-response authentication to generate data unique to the transaction.
Example: EMV payment

DDA: Dynamic Data Authentication

Card

- records, pubkey_{CARD}, sig_{BANK}\{records, pubkey_{CARD}\}
- nonce
- sig_{CARD}[card nonce, terminal nonce]
- PIN
- PIN OK/WRONG
- transaction, nonce
- MAC[transaction, nonce]

Terminal

* MAC: Message Authentication Code, computed by the card from a unique key
Example: EMV payment

DDA: Dynamic Data Authentication

YES card clone not possible: Because $\text{sig}_{\text{CARD}}\{\text{card nonce, terminal nonce}\}$ is different at every transaction.

However, card answer to PIN check is not authenticated either.

A wedge between a stolen card and terminal can pretend that the password is always correct.
Example: EMV payment

DDA: Dynamic Data Authentication

- Stolen Card
  - records, pubkey_{CARD}, sig_{BANK}\{records, pubkey_{CARD}\}
  - nonce
  - sig_{CARD}\{card nonce, terminal nonce\}

- Wedge

- Terminal
  - Wrong PIN
  - PIN OK
  - transaction, nonce
  - Invalid MAC

* MAC: Message Authentication Code, computed by the card from a unique key
Example: EMV payment

CDA: Combined DDA/Application Cryptogram Generation

Solution: Second card authentication step after PIN check.

The terminal send a message called CVMR representing the terminal view of the operation (PIN OK, PIN Wrong, signature, etc) for the card to compare with its own point of view.
Example: EMV payment

CDA: Combined Data Authentication

Card

<table>
<thead>
<tr>
<th>records, pubkey_{CARD}, sig_{BANK}{records, pubkey_{CARD}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonce</td>
</tr>
<tr>
<td>sig_{CARD}{card nonce, terminal nonce}</td>
</tr>
<tr>
<td>PIN</td>
</tr>
<tr>
<td>PIN OK/WRONG</td>
</tr>
<tr>
<td>CVMR, nonce</td>
</tr>
<tr>
<td>sig_{CARD}{CVMR, nonce}</td>
</tr>
<tr>
<td>transaction, nonce</td>
</tr>
<tr>
<td>MAC{transaction, nonce}</td>
</tr>
</tbody>
</table>

Terminal

* MAC: Message Authentication Code, computed by the card from a unique key
Example: EMV payment

Takeaway:
Do not send static auth data (e.g. unencrypted passwords)

Use challenge-response to prevent replay attacks

Make sure that authentication is verified at all steps.

For more details: